PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44

A1

(11) International Publication Number: WO 00/04445

(43) International Publication Date: 27 January 2000 (27.01.00)

(21) International Application Number: PCT/US99/16152

(22) International Filing Date: 16 July 1999 (16.07.99)

(30) Priority Data:
60/093,260 17 July 1998 (17.07.98) US

(71) Applicant (*for all designated States except US*): OMR SYSTEMS CORPORATION, INC. [US/US]; Suite 220, 101 Business Park Drive, Skillman, NJ 08558 (US).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*): FOBES, Dan [–/US]; 1609 Clydesdale Circle, Yardley, PA 19067–4110 (US). MALEY, John [US/US]; 9 Silverthorn Lane, Belle Mead, NJ 08502 (US). GLOCK, Tom [US/US]; 67 Mtn. Church Road, Hopewell, NJ 08525 (US).

(74) Agents: JIMENEZ CROWSON, Celine et al.; Rothwell, Figg, Ernst & Kurz, Suite 701 East, Columbia Square, 555 13th Street, N.W., Washington, DC 20004 (US).

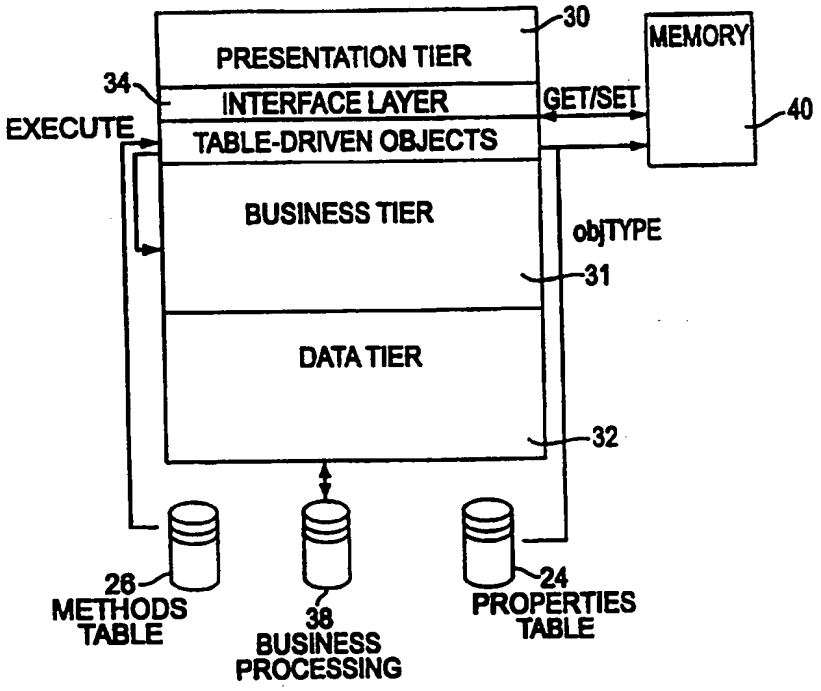
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: OBJECT ORIENTED GENERIC SCHEMA TO SUPPORT DATABASE OPERATIONS IN A MULTI–TIER ARCHITECTURE

(57) Abstract

A table–driven object for use in a presentation tier (30) of a multi–tier program to manage information for requests of types presentation and business. An object contains properties stored in a table. Objects stored in the table contain the following: "objType" property, used to dynamically configure the object; "Set(property, value)" method, used to set the value of an attribute; "Get(property, value)" method, used to get the value of an attribute; "Execute(request)" method, used to execute a method for messaging to business tier (31); "TurboExecute(request, value)", used to execute a method for messaging to business layer with reduced user input. System supported by application framework consisting of library for specific use.



OBJECT ORIENTED GENERIC SCHEMA TO SUPPORT DATABASE OPERATIONS IN A MULTI-TIER ARCHITECTURE

BACKGROUND OF THE INVENTION

This application claims priority from U.S. Provisional Application No. 60/093,260 filed July 17, 1998.

1. Field of Invention

The present invention relates to a method and system for the formation and use of a programming object whose schema is derived from relational database tables during program execution, i.e., a table-driven object, and further for applying it selected tiers of a multi-tier program.

2. Description of Related Art

Over the years, many models have been developed to aid in the construction of complex software applications. Today, object technology is a leading model for that task. Breaking down complex applications into a collection of simpler objects is a

logical step to simplifying the development and maintenance of software. However, object technology is not a panacea for every aspect of software development. Furthermore, in the areas where it does apply, the complexity and associated costs of development and maintenance can sometimes outweigh the benefits of its use.

Two popular software application design methodologies are structured programming and object-oriented programming. Structured programming requires the designer to identify the steps a program will take during execution. The goal of the design phase is to come up with a flow diagram of connected symbols such as boxes that represent processes, diamonds that represent decisions, and cylinders that represent data. The flow diagram represents the flow of execution the program will take when executed. Once the design is completed, modules are developed to implement the design. Structured programming design is appropriate to single-thread type applications (e.g., read information, if this do this else . . .).

In an object-oriented application, the designer considers the application as a collection of objects each containing data called properties, and functions called methods. Properties and methods form the

structure, or schema, of an object. Object models, similar to flow diagrams, are developed to aid in the construction of an object-oriented application. Object-oriented designs are appropriate for event-driven applications such as graphical user-interfaces (GUIs) that contain text boxes for data entry and buttons to execute business requests.

Today, many teams are required to implement and maintain most complex software. For both business and technical reasons, each team has different expertise, and each uses a different set of tools to accomplish its goals. Multi-tier architecture, usually, three-tier architecture, is a simple yet powerful way to embody an application. In the present specification, the following terms describe the tiers:

Presentation Tier:

The code that collects data from and displays information to the user. Simple code processing is done in this tier such as insuring all fields for a request are provided.

Business Tier:

The code that processes the user's requests.

Data Tier:

The code that manages the data as well as the data tier itself.

The programming team for the presentation tier needs an understanding of how to collect information from and display information to the user. The programming team for the business tier needs to understand how to accomplish business requests and enforce business rules such as security. Finally, the data tier team will need to understand the various business requests and their frequency to optimize performance in fulfilling requests.

Each team will not require the same tools since each team usually use an architecture suited for its requirements. Most data tiers are built using relational database management systems (RDBMS), not object-oriented databases (OODBMS). Database architects employ entity relationship models to design a database, and structured query language (SQL) to implement them. The business tier is typically built by business programmers who employ structured design techniques to design an application, and a high-level programming language such as COBOL to implement it. Languages that have built-in support for objects such as C++ can perform these operations, but the simplicity of a high-level programming language such as COBOL,

allow programmers to focus on business rules, not implementation details.

The presentation tier is a complex layer of event-driven software that is controlled by the user using the I/O devices 5 of the computer. The presentation tier is usually composed of a form with graphical controls such as text boxes for data entry and buttons for executing a request. The presentation tier benefits most from object technology because the forms and controls the user works with can be represented as objects. Here, object-oriented programmers design interfaces using object-oriented models such as Rumbaugh's Object Modeling Technique (OMT), and an object-oriented language such as C++ is used for the implementation.

The added power that comes with an object-oriented design has two drawbacks - complexity and cost. These drawbacks apply to both the initial implementation and the maintenance that follows. This is because languages that directly support objects, such as C++, do so through the use of special programming language structures. These structures enable a programmer to define an object's schema (e.g., properties and methods). As an object matures, its schema inevitably becomes more complex (e.g., more properties and methods

are added, more code for each is added). Additionally, experienced programmers are trained to recognize common objects, then break them down into a series of objects that share code with each other using an object-oriented technique called inheritance. It is also common to have one object contain other objects.

SUMMARY OF THE INVENTION

It is therefore a general advantage of the present invention that a method and apparatus are provided having flexibility in creation of programming objects.

It is a more particular advantage of the present invention that a method and apparatus are provided in which a programming object is created whose target schema is derived from database tables in a database, with a properties table and a methods table to implement the target schema.

It is another advantage of the present invention that a method and apparatus of the type described are provided in which objects with different schemes can be created dynamically from one generic schema.

It is a further particular benefit of the present invention in that a method and apparatus are provided in which one instance of a table-driven object can mutate dynamically from one run-time schema to another.

It is also a particular benefit of the present invention in that a method and apparatus are provided in which a library of complex objects, each targeted for a specific use, can be reduced to one object and its supporting tables, reducing both costs and complexity of development and maintenance.

Briefly stated, in accordance with the present invention a method and apparatus are provided for creating a programming object using database tables - a table-driven object. A table-driven object is an object with a generic schema, whose target schema is derived from database tables, not the programming language it was implemented in. It requires an object-oriented language which is used to implement the generic schema, and a database with a properties table and a methods table to implement the target schema. One instance of a generic schema is composed of one property and four methods:

1. objType property.
2. Set(property, value) method
3. Get(property, value) method
4. Execute(request) method
5. turboExecute(request, value) method

The objType property is used to configure an object, and it does this by dynamically reading the

appropriate properties from a database into a memory device. The objType property will determine how many properties are included in one object. Set(property, value) provides the mechanism to set the value of a virtual property, Get(property, value) provides the mechanism to retrieve the value of a virtual property, and Execute(request) is used to communicate a business request to the business tier using the object's properties. The turboExecute method is a mechanism for quick execution of a business request whereby a subset of property values are provided, while others are defaulted.

BRIEF DESCRIPTION OF THE DRAWINGS

The means by which the foregoing objects and features of the invention are achieved are pointed out in the claims forming the concluding portion of the specification. The invention, both as to its organization and manner of operation may be further understood by reference to the following description taken in connection with the following drawings.

Of the drawings:

Figure 1 is a hardware diagram of a computer system in which the present invention is embodied;

Figure 2 illustrates an example properties table that contains the properties of an object and each property's attributes;

Figure 3 illustrates an example methods table for further defining a table-driven object;

Figure 4 is a block diagrammatic representation of a portion of the present system illustrating a three-tier architecture with tables and memories in which the tables are located;

Figure 5 is a flow diagram detail of Figure 4 illustrating the sequence of steps performed as a result of setting the table-driven object's objType;

Figure 6 is a flow diagram detail of Figure 4 illustrating the sequence of steps performed as a result of calling the table-driven object's Set Method;

Figure 7 is a flow diagram detail of Figure 4 illustrating the sequence of steps performed as a result of calling the table-driven object's Get Method;

Figure 8 is a flow diagram detail of Figure 4 illustrating the sequence of steps performed as a result of calling the table-driven object's Execute Method;

Figure 9 is a flow diagram detail of Figure 4 illustrating an example sequence of steps to accomplish a turbo business request;

Figure 10 is sample code that illustrates the creation of a table driven object, setting its properties, executing a business request, then reading the results back; and

Figure 11 is sample code that illustrates the creation of a table driven object, executing a turbo business request, then reading the results back.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to Figure 1, the method of the present invention may be practiced on and embodied in a computer terminal 1. The computer terminal 1 includes a computer 2 having a memory 3, central processing unit 4, and input/output unit 5. Input/output devices including a monitor 10, keyboard 11 and mouse 12 are each coupled to the input/output unit 5. Additionally, the input/output unit 5 communicates with a database 20 which may be accessed via a network 22 or in other known ways.

In the present description, Figures 2 and 3 illustrate tables used in forming a table-driven object. Figure 4 is an illustration of three-tier architecture illustrating tables and memories and their location. Figure 5 is an illustration of the method for setting the object type. Figures 6 and 7 are

illustrations of the steps performed in response to calling a Set method and a Get method respectively. Figure 8 illustrates the response to calling the Execute method and Figure 9 is a flow diagram illustrating the sequence of step for turbo business requests.

More specifically, Figure 2 illustrates one example of a properties table 24. In the presentation tier, an object's properties manage information for presentation and business requests. The table-driven object accomplishes this by having all of its properties in a table and associating attributes to each property. The properties and associated attributes are loaded into memory 64 (Figure 5) as a result of setting the objType property, and property values are set and retrieved using Set and Get methods as shown in Figures 5, 6, and 7 respectively. Setting the objType type causes the appropriate properties to be loaded from the properties table 24 into memory, and any changes to an object's schema is done in the table 24 using known methods for adding, changing, and removing rows in a table, not in the programming language. This includes adding new properties (add a new row to table), deleting existing properties (delete a row from the table), and changing the attributes of a

property(update a row in a table). Changes to objects are available immediately by simply setting the objType. This is in contrast to coding changes directly into the object, rebuilding the object, deleting the older version of that object, and installing the newer version.

A table-driven object is an object with a generic schema implemented in a programming language, and a target schema derived from database tables, not a programming language. It requires an object-oriented language which is used to implement the generic schema, and a database 20 with a properties table 24 and a methods table 26 to implement the target schema. One example of a preferred generic schema is composed of one property and 4 methods to accomplish this:

1. objType property.
2. Set(property, value) method
3. Get(property, value) method
4. Execute(request) method
5. turboExecute(request, value) method

Table 24 in Figure 2 is one example of how the objType property is set. Different properties, each of which will be in the database 20 (Figure 1), are specified for the object of the present example. In one example in accordance with the invention, consider a

trade entry application whose users work with products and instruments. Such an application could contain a Spot product that facilities the trade of one instrument for another (e.g., U.S. Dollars for Japanese Yen), and a Deposit product that facilitates the deposit of some instrument such as U.S. Dollars over a specified time. Traders will enter product data then make a request such as "Edit" which checks the deal (requested trade) for validity, and "Commit" which writes the trade. "Edit" and "commit" are the methods in the present example, further described with respect to Figure 3, 26.

In the example, the properties include "User name," "Trade ID," "Company," "Changed" and "Notes" (Figure 2, 24). These property names will be given property IDs 1-5 respectively. In each case the request Id for edit will be 1. Default values for a subset of properties are entered for provision in the "turboExecute" mode. In the present examples, the values will be null, or blank in the case of the user name, trade ID, change and notes. The company name will be defaulted in the case of a Spot object. The table 24 also specifies that a user can request a "turboExecute" form of communication further described below.

With the properties in memory, there are 2 options for iterating through all of an object's properties. Using known methods supporting the structure implemented, nodes in memory 64 (Figure 5) can be "walked." Alternatively, the table 24 (Figure 2) can be queried such that all rows for a given object name/type are returned. From these rows, the id's can be retrieved and used as input to the Get method of Figure 7. This methodology can be used to build argument lists for business processing. By adding a request identifier and a parameter number column to the properties table 24, properties can be dynamically mapped to a business request. For any business request, the object has all the information needed to gather the arguments for a request allowing it to perform the call. This is explained in more detail below.

The implementation of virtual properties must scale. Complex objects will have a large number of properties, and the user must be able to set them or retrieve them without any noticeable performance penalty. If there are 1,000 properties, assigning a value to the $1,000^{th}$ property should not take any more time than assigning a value to the first. This argues for an indexing scheme in which the property name is

indexed. Since properties reside in the database 20 (Figure 1), one option is to have their values reside there, effectively making them persistent. Persistent presentation objects address the scaling issue mentioned above, however, the overhead of this for every access is usually too great for an object that changes as frequently as a presentation object. For performance reasons, the presentation object's properties should be read from the database 20 and stored in memory using known organizations that support quick updates and retrievals based on an index (e.g., B-tree), where the index is the property name.

Figure 3 illustrates an example of a methods table 26 associated with the table-driven object of Figure 2. Execute is the mechanism that uses table 26 to perform a business request. Additionally, TurboExecute is a mechanism for quick execution of a business request whereby a subset of object attributes are provided with property values while others are set to default values in accordance with the defaults column in table 24 of Figure 2. The object of the present example will also include the methods "Edit" and "Commit" as shown in table 26 in Figure 3. The "Edit" method is for allowing a user to submit a trade for checking by the

system. The "Commit" method is for actual ordering of a trade.

A preferred embodiment of the multiple tier architecture is illustrated in Figure 4. The computer 2 (Figure 1) has a presentation tier 30, business tier 31 and data tier 32. The tiers 30-32 are stored in the computer 2 in a known manner. An interface layer 34 and the table-driven objects are included in the presentation tier 30. Persistent data required for business processing is located on a disk 38 and is coupled to and interfaces with data tier 32. Persistent data required for the invention is located in methods table 26 and properties table 24. Memory 40 is required to temporarily hold data to support the methods of table-driven objects.

In operation, as illustrated in Figure 5, the object's objType is supplied as seen at box 60. At box 61, the objectType is retrieved from table 24 (Figure 4) using known methods (e.g., select * from Properties where Properties.objName=objType). If none appear, that is no rows are returned from the query, as indicated at box 62, the command is returned with a message. When a proper object type is detected, as indicated at box 63, all rows from the properties table 24 residing on disk 65 are read into the memory device 64. Preferably each

node in the tree of the memory device 64 should be indexed by the property ID using known methods and each node should contain all columns of the properties table 24 in addition to a field that will hold the value.

In order to provide values to be used for the object, the Set method is called (Figure 6). The proper ID is supplied at box 70, and as seen at box 71, known searching techniques are applied to the memory 64 (Figure 5) to find a node representing the property queried. If a node with this ID is not located in memory 64, then at box 72 an appropriate message is returned. If the search results in a found node for that particular ID, then operation proceeds to the box 73, where using the node found in memory 64 from the search in 71, the value for this property is changed within the node to the value passed in.

The Get method is illustrated in Figure 7. A property ID is supplied and read as indicated at box 80. Then at box 81, known searching techniques are applied to the memory device 64 of Figure 5 to produce a search result. If a node with this ID is not located in memory 64, then at box 82 an appropriate message is returned. If the search results in a found node for that particular ID, then operation proceeds to the box

83, where the value for this property is retrieved from within the node and returned.

Business requests are executed by the business tier 31 (Figure 4), not the presentation tier 30. The presentation tier 30 collects the information required for the request, then passes selected information to the appropriate module in the business tier 31 for processing. Referring now to Figure 8, the Execute method in the table-driven object simplifies all business requests by centralizing them to one method with an argument that identifies the request. The Execute method will collect the arguments required for the request using the request ID and parameters column in the properties table 24 (Figure 2), and call the appropriate business processing module as shown in Figure 8. At box 90, an ID is supplied and read. If this is not a valid ID as measured at box 91, then at box 92, the request is returned with an appropriate message. A valid request initiates progress to box 93 at which the business program arguments are built by retrieving the parameters for that request sorted by parameter number as indicated in the property table 24 and subsequently the values for those parameters are retrieved from the memory device 64 (Figure 5). At box 94, the module name is retrieved from the methods table

26 (Figure 3) and the request is executed by calling the business module associated with that request with the specified parameters.

The invention provides, e.g., Figure 4, for dynamic methods by employing a methods table 26 that contains the requests, a request identifier, and the associated business processing module name, and adds a column to the properties table 24 that contains the request identifier. Specifically, the invention allows for an interface 34 to query the properties table 24 for a particular objectType and determine the business requests it can perform by checking for those that have non-zero arguments. It can then use the methods table 26 to display the requests available for the selected object type to the user. When the user instructs the interface 34 to perform one of the business requests, the object looks up the identifier of the request being made, queries the properties table 24 for all properties associated with that request, gathers the values for those properties using the Get method of Figure 7, and executes the appropriate business processing module which it retrieves from the methods table 26 using the data from the properties as parameters to the call.

Sample programming code for creating a table-driven object, executing a business request and retrieving the results is contained in Figure 10.

The table-driven presentation object also provides for business requests that can be entered by the user in one line - a turbo business request. Some business requests require many pieces of information, and users may find these too time consuming - especially if most of the information is constant. A turbo business request allows a user to enter a subset of the parameters on one line. The turbo business request is accomplished by adding the defaults and turbo request columns to the properties table 24. The user enters one line of information which corresponds to a subset of properties for a particular table-driven object. The sequence of steps in a preferred embodiment for turbo business requests is shown in Figure 9.

Specifically, a turbo request is submitted by the user and received at box 100. Here the request is "Spot JDOE USD DEM 1000". Using known parsing techniques, the first word is identified as spot and subsequently, a spot object is created. At box 100, properties for a table-driven object representing a spot exists in the memory device 64 (Figure 5) by creating a table-driven object and setting its object

type equal to "Spot." This creation of the spot comprises the "objType" property of Figure 5. As seen at box 101, using the properties table 24, default values for a "Spot" object are moved into the memory device 64. At box 102, table 24 is used to obtain the default values for a "Spot" object. The remaining portion of the request, "JDOE USD DEM 1000" is parsed according to known techniques and using the information in table 24, values for username and other fields are set. And as indicated at box 103, the Execute method of Figure 8 is performed to provide a complete object for performance of an edit or a trade. Upon some user action, such as a hitting an enter key or a button press, the table-driven object is instructed to execute the turboExecute method.

The turboExecute method first moves the default values for the business request parameters into the memory 64 (Figure 5). It then parses the line entered using the turbo request column of the properties table 24 to determine the line format, and effectively fills in the values of a subset of properties. Finally, it calls the Execute method. Variations such as per user defaults and turbo line formats can be accomplished by adding a user column to the properties table 24, or breaking the properties table 24 into two tables - one

for the properties, object name, and business request information, and one for the user's defaults and turbo format.

In this example, the application is built on a three tier architecture where the business tier 31 (Figure 4) is implemented in COBOL and the presentation tier 30 is implemented in Microsoft's Visual Basic. The COBOL tier has an interface program that accepts a module name and arguments from the presentation tier 30, and calls the appropriate COBOL module with those parameters to execute that request. Errors and warnings are returned to the presentation tier 30 through the arguments (e.g., a status and message property). The business tier 31 is physically decoupled from the presentation tier 30, and requests between the presentation tier 30 and the business tier 31s are accomplished using messaging middleware (e.g., a message containing the request and the parameters is sent over a network to a receiving program that executes it). The presentation tier 30 includes two layers - an interface layer 34 and a table-driven object layer 36.

The interface layer 34 is a collection of forms written in Visual Basic that gather data from the user for a particular product (e.g., Spot). Once the user

enters the data for a spot, a request such as edit, which checks the trade, is executed. Upon receipt of a business request from the presentation tier (i.e., a button click), the interface code builds a spot object by initializing a table-driven object and setting its type to spot.

Consequently, the table-driven object loads all spot properties from the properties table 24 into the memory 64.

The interface code then sets the values of the properties from the form and possibly other values it obtains from a database itself, or it calculates, using the Set method of the table-driven object (Figure 6). After setting all of its properties values, the table-driven object is instructed to call the Execute method (Figure 8) with the Edit parameter by the interface layer 34.

The calling of a remote business processing program in Box 94 (Figure 8) is accomplished via messaging middleware, where the message contains the business interface program module name and parameters it requires. After sending the request to the appropriate business module, the interface layer 34 (Figure 4) blocks until the business module returns. Upon return, Execute refreshes its properties using the

parameters returned from the business tier 31, and returns control back to the interface 34. The interface 34 uses the Get method illustrated in Figure 7 to examine fields that represent errors and warnings, and flushes selected properties of the spot object to the form for user display.

Sample code for creating a table-driven object, executing a turbo business request and retrieving the results is shown in Figure 11.

The invention reduces costs of development and maintenance by providing an object with a simple interface that supports many types of objects. There is a reduction in the size of an application built with these objects, the object library itself, and the amount of overall memory required by the application. Specifically, given an application built employing standard object technology and one based on the table-drive object model of this invention, some preliminary results in the context of the application discussed in the specification are as follows:

Application: 19% decrease

Object Library: 59% decrease

RAM: 16% decrease

Also, moving from one interface to another is much simpler. For example, a browser version of the above

application may be accomplished by recoding the interface in HTML and having it create table-drive objects, set properties, and execute business requests as was done with the application employing Visual Basic.

In applications where the presentation tier 30 (Figure 4) performs complex business processing, it may be advantageous to employ a layer of standard objects that receive requests from table-driven objects in accordance with the invention. This would permit inheritance of one object from another and one object to contain another.

It should be recognized that every application can benefit from applying the table-driven object of this invention to the presentation tier 30 of the application. Legacy-type systems, in particular, however, benefit significantly from the table-design object of the invention. For example, without any code rewrites, legacy applications and their data become accessible to the latest presentation technology. New presentation technology becomes more of an add-on than a rewrite. For legacy applications the method of a preferred embodiment of this invention includes the following steps:

1. Break down the target application into 3 tiers.
2. Define the interface portion of the presentation tier (forms, etc.).
3. Identify the business requests and their parameters required for the business tier. If necessary, create one or more business request processing programs in the business tier.
4. Build the table for the table-driven object to accomplish the requests of 3 and the interface of 2.
5. Implement the interface, create and populate the table-drive object, and send the requests in.

CLAIMS

1. A method for forming a table-driven object for use in a multi-tier architecture comprising the steps of forming a generic schema in an object oriented computer language and providing a database with a properties table and a methods table for implementing a target schema, the generic schema comprising at least one property and a plurality of methods, the property defining a preselected group of properties to be read from a database into a memory device housing a target schema, at least one said method for determining values to be assigned to and at least one said method for determining values to be retrieved from attributes in said target object and at least one said method for communicating with a business tier of said architecture for utilizing the target object in a business request;

invoking said at least one method for determining attribute values and storing attributes of said target object in the memory and invoking said at least one method for communicating with said business tier for providing said object thereto.

2. The method according to Claim 1 further comprising the step of providing attributes available in said database and selectively altering properties of

said generic object for providing for dynamic mutation of object attributes.

3. The method of Claim 1 wherein the step of providing at least one method for updating attribute values comprises providing a Set method and providing means for responding to invocation of said method for providing a value for each property of the object.

4. A method according to Claim 1 wherein the step of providing at least one method for determining an attribute value comprises a Get method and further comprising the step of providing means for responding to invocation of said get method for returning to the object a corresponding value determined by said get method.

5. A method according to Claim 1 wherein said at least one method for communicating comprises an Execute method for communicating a business request to the business tier including the object properties.

6. A method according to Claim 1 where said at least one method for communicating comprises a turbo execute method and the step of communicating further

comprises communicating a limited number of attribute values determined in accordance with the properties table and for communicating default values for preselected attributes.

7. An apparatus comprising means for storing a target object for provision to a presentation tier in a multi-tier architecture, said means comprising an object memory for storing a target object, a database storing values for association with attributes of a target object, tables for storing methods and attributes for target objects, means for storing a generic object and means for invoking said generic object, and means for loading said object memory in response to reading of said tables as commanded in accordance with said generic object, said target object being accessible to response to further commands.

8. An apparatus according to Claim 7 further comprising means for commanding coupling of said stored object to a business process in a multi-tier architecture for handling user requests in a presentation tier and data manipulation in a business tier, and means storing an object having attributes and methods, means for communication with said object to

provide values for attributes therein in the user tier and means for coupling said object as an element in business requests in said business tier.

9. A method for performing a transaction comprising the steps of storing a generic object, said object comprising at least a set of properties and methods for communicating with a presentation layer for determining object attribute values and for communicating with a business layer for performing a transaction; commanding construction of a target object; accessing values from a database for provision to said target object memory; and coupling the attributes of said target object in a transaction.

10. An apparatus for performing a transaction comprising a presentation tier which determines object attribute values, a business tier which performs a transaction and means for commanding communication between said business tier and said presentation tier, means for commanding construction of a target object, said means for commanding communication with said presentation tier comprising means for accessing values from a database for a provision to said target object memory and wherein said means for communicating with

said business tier comprises means for coupling the attributes of said target object in a transaction.

11. An apparatus in accordance with Claim 10 further comprising means for dynamically mutating the target object.

12. An apparatus in accordance with Claim 10 wherein said means for dynamically mutating comprises means for selecting at least one differing attribute from said database for inclusion in said target object.

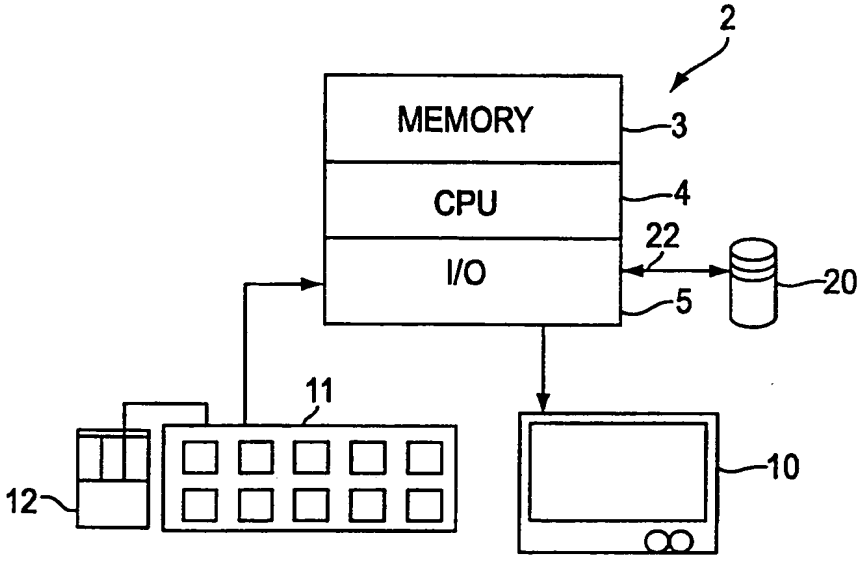


FIG. 1

24

PROPERTY NAME	PROPERTY ID	OBJECT NAME	REQUEST ID	PARAM NUMBER	DEFAULTS	TURBO REQUEST
USERNAME	1	SPOT	1	1	NULL	1
TRADE ID	2	SPOT	1	0	NULL	0
COMPANY	3	SPOT	1	2	BB	0
CHANGED	4	SPOT	1	0	NULL	0
NOTES	5	SPOT	1	3	NULL	0

FIG. 2

26

REQUEST NAME	REQUEST IDENTIFIER	REQUEST MODULE
EDIT	1	cblEDIT
COMMIT	2	cblCOMMIT

FIG. 3

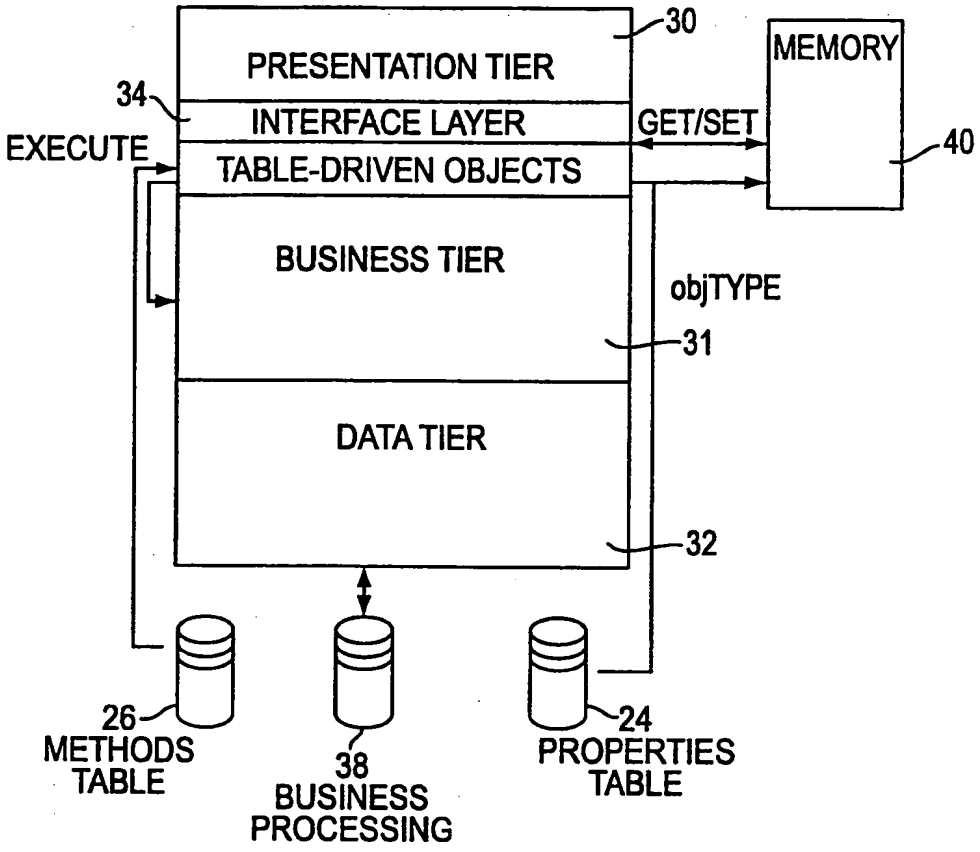

PRESENTATION TIER
30
MEMORY
34
INTERFACE LAYER
GET/SET
EXECUTE
TABLE-DRIVEN OBJECTS
40
BUSINESS TIER
objTYPE
31
DATA TIER
32
26
METHODS TABLE
38
BUSINESS PROCESSING
24
PROPERTIES TABLE

FIG. 4

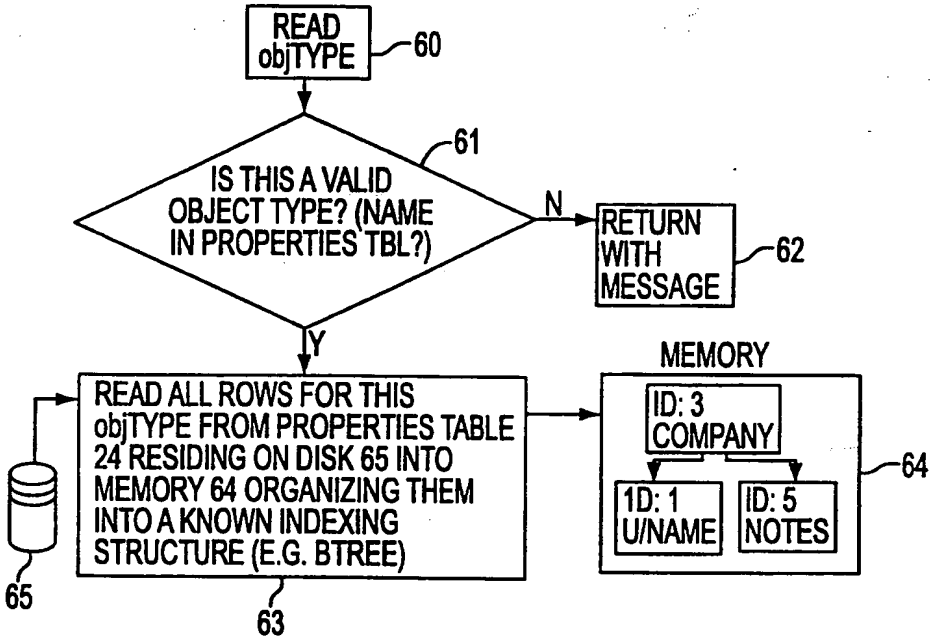

READ objTYPE
60
IS THIS A VALID OBJECT TYPE? (NAME IN PROPERTIES TBL?)
61
N
RETURN WITH MESSAGE
62
Y
READ ALL ROWS FOR THIS objTYPE FROM PROPERTIES TABLE 24 RESIDING ON DISK 65 INTO MEMORY 64 ORGANIZING THEM INTO A KNOWN INDEXING STRUCTURE (E.G. BTREE)
63
65
MEMORY
ID: 3 COMPANY
1D: 1 U/NAME
ID: 5 NOTES
64

FIG. 5

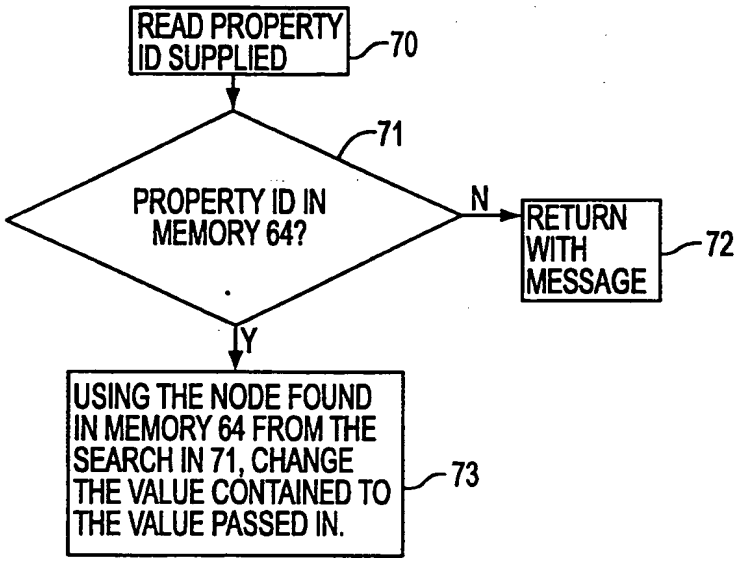

READ PROPERTY ID SUPPLIED
70
71
PROPERTY ID IN MEMORY 64?
N
RETURN WITH MESSAGE
72
Y
USING THE NODE FOUND IN MEMORY 64 FROM THE SEARCH IN 71, CHANGE THE VALUE CONTAINED TO THE VALUE PASSED IN.
73

FIG. 6

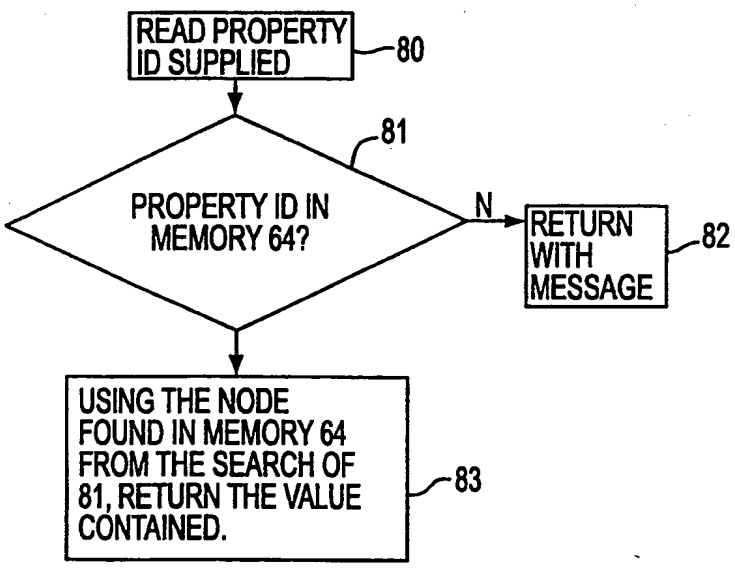

READ PROPERTY ID SUPPLIED
80
81
PROPERTY ID IN MEMORY 64?
N
RETURN WITH MESSAGE
82
USING THE NODE FOUND IN MEMORY 64 FROM THE SEARCH OF 81, RETURN THE VALUE CONTAINED.
83

FIG. 7

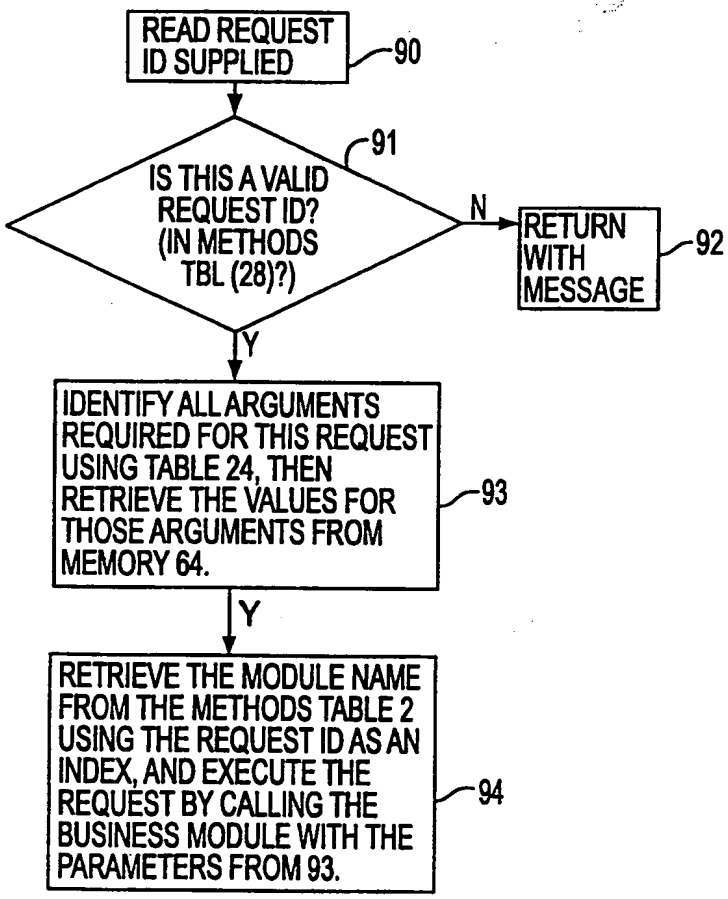

READ REQUEST ID SUPPLIED
90
IS THIS A VALID REQUEST ID? (IN METHODS TBL (28)?)
91
N
RETURN WITH MESSAGE
92
Y
IDENTIFY ALL ARGUMENTS REQUIRED FOR THIS REQUEST USING TABLE 24, THEN RETRIEVE THE VALUES FOR THOSE ARGUMENTS FROM MEMORY 64.
93
Y
RETRIEVE THE MODULE NAME FROM THE METHODS TABLE 2 USING THE REQUEST ID AS AN INDEX, AND EXECUTE THE REQUEST BY CALLING THE BUSINESS MODULE WITH THE PARAMETERS FROM 93.
94

FIG. 8

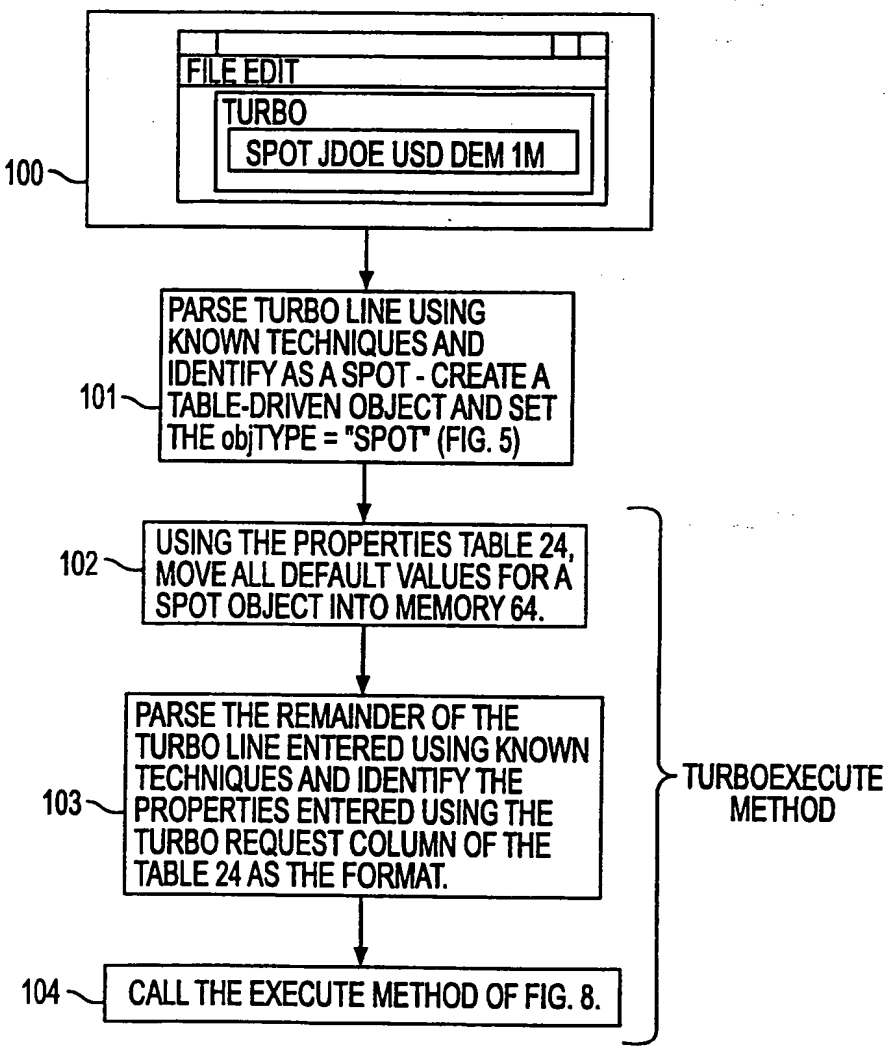

FILE EDIT
TURBO
SPOT JDOE USD DEM 1M
100
PARSE TURBO LINE USING KNOWN TECHNIQUES AND IDENTIFY AS A SPOT - CREATE A TABLE-DRIVEN OBJECT AND SET THE objTYPE = "SPOT" (FIG. 5)
101
USING THE PROPERTIES TABLE 24, MOVE ALL DEFAULT VALUES FOR A SPOT OBJECT INTO MEMORY 64.
102
PARSE THE REMAINDER OF THE TURBO LINE ENTERED USING KNOWN TECHNIQUES AND IDENTIFY THE PROPERTIES ENTERED USING THE TURBO REQUEST COLUMN OF THE TABLE 24 AS THE FORMAT.
103
CALL THE EXECUTE METHOD OF FIG. 8.
104
TURBOEXECUTE METHOD

FIG. 9

```
// declare the table-driven object
TDObject myTDObject;
int errorCount;

// create a Spot product object by loading all rows from table 24 with Spot as
// Object Name into a memory device similar to Figure 5.
myTDObject.objType = "Spot";

// set username property
myTDObject.Set("Username","JDOE");
// other properties required for a business request to follow
// such as instruments to trade, how much, who to trade with, etc.

// execute an edit request - check the trade
myTDObject.Execute("Edit");

// check for errors
myTDObject.Get("ErrorCount",&errorCount);
if errorCount > 0 then {
        // error handler
}
```

FIG. 10

```
// declare the table-driven object
TDObject myTDObject;
int errorCount;

// create a Spot product object by loading all rows from table 24 with Spot as
// Object Name into a memory device similar to Figure 5.
myTDObject.objType = "Spot";

// execute an edit request - check the trade
myTDObject.turboExecute("Edit", "JDOE", etc.);

// check for errors
myTDObject.Get("ErrorCount",&errorCount);
if errorCount > 0 then {
        // error handler
}
```

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/16152

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 9/44
US CL : 707/100,101,103,104

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/100,101,103,104
395/701,702,703,704,705,706,707,708,712

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Please See Extra Sheet.

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

DIALOG - Inspec, Compsci, Compulab, Compendex
APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	Roundtable on Object-Oriented Technologies - Object Oriented DBMS 1989	1-12
Y	Principles of Object-Oriented Analysis and Design James Martin 1993 chapters 12, 13,15,16, 19,21 22	1-12
Y	New Objectivity Version Adds SQL	1-12

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
26 AUGUST 1999

Date of mailing of the international search report
22 OCT 1999

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
TODD INGBERG James R. Matthews
Telephone No. (703) 305-3800

Form PCT/ISA/210 (second sheet)(July 1992)★